# A scalable architecture for online anomaly detection of WLCG batch jobs



2016 J. Phys.: Conf. Ser. 762 012002

(http://iopscience.iop.org/1742-6596/762/1/012002)

# A scalable architecture for online anomaly detection of WLCG batch jobs

**E Kuehn, M Fischer, M Giffels, C Jung, A Petzold**

Karlsruhe Institute of Technology, Steinbuch Centre for Computing,
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany

E-mail: {eileen.kuehn, max.fischer, manuel.giffels, christopher.jung, andreas.petzold}@kit.edu

**Abstract.** For data centres it is increasingly important to monitor the network usage, and learn from network usage patterns. Especially configuration issues or misbehaving batch jobs preventing a smooth operation need to be detected as early as possible. At the GridKa data and computing centre we therefore operate a tool BPNetMon for monitoring traffic data and characteristics of WLCG batch jobs and pilots locally on different worker nodes. On the one hand local information itself are not sufficient to detect anomalies for several reasons, e.g. the underlying job distribution on a single worker node might change or there might be a local misconfiguration. On the other hand a centralised anomaly detection approach does not scale regarding network communication as well as computational costs. We therefore propose a scalable architecture based on concepts of a super-peer network.

## 1. Introduction

Misbehaving batch jobs can be a notable issue in batch systems as they can negatively affect other batch jobs. Especially negative effects on network traffic rates have potential to increase the walltime enormously. E.g., batch jobs report their results and therefore rely on the network. However, existing monitoring solutions do not provide network metrics distinguished by batch job. Therefore, a network traffic monitoring tool BPNetMon with process level resolution has been developed and put into operation at the GridKa data and computing centre [1]. It monitors information on UNIX process level with a focus on network traffic. By tracking the call hierarchy of single processes started by a batch job, information can be aggregated on batch job level.

Especially in the field of high energy physics the tool is useful to gain insights into the workflows of pilots [2]. Pilots form a virtual overlay batch system [3]. They load batch jobs, following *payloads*, from an external batch system queue. This happens independently from the underlying batch system. While this improves scheduling for the users, the information available to the underlying system is drastically reduced.

These information can also be monitored with BPNetMon. As all monitoring information are reported in a tree data structure, relevant information on payload level can be extracted by identifying the parent process. With access to details on payload level, we now want to introduce an online anomaly detection, that enables a near real time recognition of anomalous payloads. A working anomaly detection on payload level ensures an equal and undisturbed usage of resources for all users of the batch system.

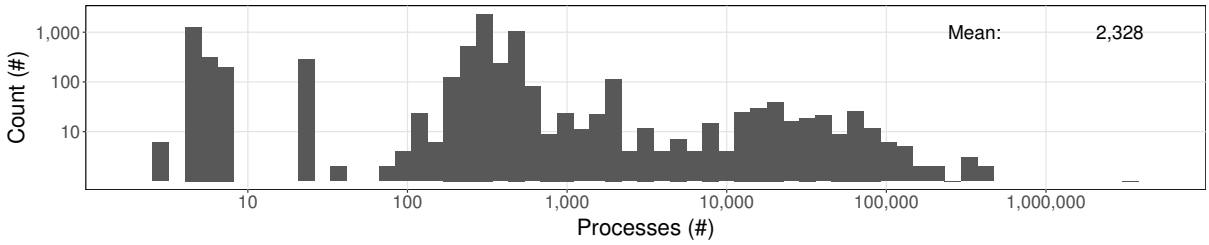


**Figure 1.** Number of processes to consider when analysing CMS payloads. The mean number of processes is around 2,300 whereas the maximum number recorded is up to 3,600,000 processes. This requires a good scalability. In addition, there is evidence for clusters of similar payloads based on their process count.

## 2. Problem description

The following discussions and assumptions are mainly based on monitoring data from the CMS collaboration [4]. In addition, only pilots are considered. Statistics are based on data that has been collected offline on one of the 600 worker nodes of the GridKa data and computing centre. The selected worker node offers 24 job slots. The data has been collected over a period of 12 months from July 2014 to June 2015.

### *2.1. Anomaly detection*

The basis to detect anomalous batch jobs is knowledge about normal behaviour. As normal behaviour is unknown, it first needs to be learned. To define normal behaviour we consider majority of batch jobs as normal. Only a small fraction are outliers. If this holds true, normal behaviour can be defined based on natural breaks regarding the distribution of process count per payload. Given the dataset of payloads for CMS collaboration of one worker node, we therefore distinguish five groups of payloads describing normal behaviour (compare Figure 1). For each of those groups, the average payload is calculated. This average is called a *prototype* in the following.

A prototype is the representative for the whole group and can be used to measure the distance to a running batch job. Based on that distance, one can decide if the batch job belongs to that group, and therefore behaves normally. Otherwise it can be treated as an outlier.

### *2.2. Distance measurement for trees*

As the batch job monitoring works on process level, we deal with tree data structures. To measure distances between trees there exist many solutions in literature, e.g. the tree edit distance [5]. The distance is given by the number of operations required to transform one tree $T_1$ with $n$ nodes into another tree $T_2$ with $m$ nodes. Operations permitted are the deletion, addition, and relabelling of nodes. State of the art algorithms of tree edit distance require $O(n^3)$ time and $O(nm)$ space resources [6].

However, these algorithms are designed to measure exact distances for static trees. Our use case requires an efficient solution that works on an event stream $E_{\text{job}} = \langle e_1, \ldots, e_n \rangle$ with bounded memory and CPU resources. The events given by $E_{\text{job}}$ describe a tree $T_{\text{job}} = \langle T_1, \ldots, T_n \rangle$ of a batch job that is dynamic over time. Distances repeatedly need to be calculated to a given set of prototype trees $P = \{T_{\text{prototype},1}, \ldots, T_{\text{prototype},i}\}$. Complexity for exact distance measurements for $i$ prototypes extends to $O(in^3)$ time and $O(inm)$ space requirements. Therefore an incremental distance measurement is preferred.

*2.3. Exemplary demonstration*
Let us consider to perform anomaly detection for an average payload with $2,300$ processes for 5 groups of normal behaviour based on tree edit distance. We first need to measure distances to each of the 5 prototypes. We require resources for $5 \cdot 2,300 \cdot 2,300 = 26,450,000$ objects in average for one event at a time.

As there are up to 24 job slots per worker node, we need to consider up to $24 \cdot 26,450,000 = 634,800,000$ objects. In addition, we still need to store the data per process per payload and prototype. On average, each process requires 38 B of memory. This requires a total memory of 2.00 MB for payload as well as 0.42 MB for prototype storage and 2.36 GB for distance calculations.

Taking into account, that memory should not exceed 4.00 GB one out of the 24 payloads with $40,400$ processes already exceeds the given limits. This is the case for 1.10 % of the payloads on the given worker node. Hence, it is practically impossible that limits are not exceeded when running on 600 worker nodes with tree edit distance.

*2.4. Requirements*
In summary, different aspects need to be considered to put a scalable architecture for online anomaly detection into practice. First, algorithmic complexity to determine distances between trees is of interest. So far, no sufficient scalable algorithm has been identified. Currently we are testing an incremental distance measurement approach that looks promising. It is based on lossy compression of trees and the usage of hash tables.

Second, the learning of prototypes needs to be considered. This can be done offline on a central node before distributing the prototypes to the different worker nodes. But it might also be considered to have different prototypes per worker node.

Most importantly the uninterrupted operation of the batch system is key. Both, a low communication overhead as well as low memory consumption and CPU load on worker nodes are crucial.

## 3. A scalable architecture for online anomaly detection
Currently the monitoring tool BPNetMon is deployed on two racks, each consisting of 32 worker nodes. Each worker node provides 24 job slots. The tool is operated as a sensor that transfers monitoring events to a central instance for further offline analysis and prototype learning. An online anomaly detection has not been deployed yet. Following, we will discuss feasible options for deployment and summarise advantages and disadvantages of the presented architectures.

*3.1. Centralised systems*
The initial situation enables a centralised system with one central server. The main task of the central server is the anomaly detection. Necessary calculations do therefore not interfere with batch jobs that are running on the worker nodes. We also benefit from data consistency, coherence, and a centralised prototype model. However, we expect very high CPU load as well as memory demands on that central server. The current setup already requires the parallel analysis of $64 \cdot 24 = 1,536$ payloads. Based on a tree edit distance, the distance calculations themselves require around 151 GB of memory. A scaling to the whole GridKa data and computing centre can therefore not be achieved.

Based on measurements for several months we simulated the network overhead for such a centralised system. We want to ensure that the tool generated traffic does not exceed 3.00 % of traffic that is induced by batch jobs and system relevant processes. The simulation shows that the tool generated traffic exceeds the threshold (compare Figure 2). Actually the simulation showed, that the threshold is exceeded in 10.44 % of all monitoring intervals. To ensure scalability, a preselection of data should be performed on the different worker nodes.

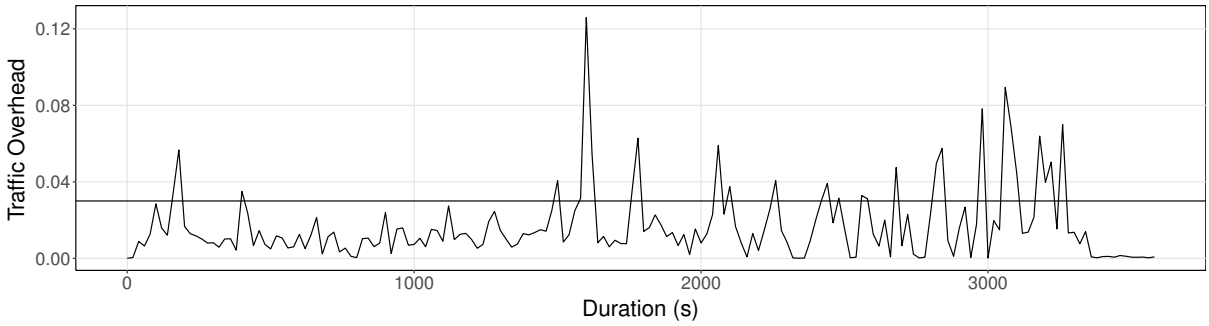


**Figure 2.** Simulated traffic overhead for a centralised system. The tool generated traffic should not exceed 3.00 % of batch job induced traffic. As shown, the generated traffic exceeds the threshold. In addition, the simulation showed that this is the case in 10.44 % of all monitored time intervals. To minimise this overhead, a preselection of data is required.

The setup consisted of 10 worker nodes and one central instance. The different hosts were connected by 1 Gbit/s full-duplex ethernet connections. Randomly a replay from log files was started to induce network activity for the different worker nodes. We first considered base traffic for batch jobs and system relevant processes. In addition we simulated the events produced by BPNetMon and transfered those to the central host.

### *3.2. Hierarchical systems*

To tackle the disadvantages of centralised systems we can introduce additional layers of local servers. Each local server groups several worker nodes. Worker nodes still send monitoring information to their associated server. The local servers collect information from a group of worker nodes and perform a preselection of received data. Selected data are then send to the central server. The central server performs anomaly detection on a reduced set of data. Hence, a logical hierarchy is implemented that allows further scalability. As information flow is retained, data consistency, coherence, and a centralised prototype model are ensured.

A hierarchical system offers improved scalability over centralised system but still requires additional hardware for servers. However, chances for a failure of the whole anomaly detection process are reduced but the detection may still fail for groups of worker nodes.

### *3.3. Peer-to-peer systems*

A more flexible architecture regarding the failure of the whole system or groups of worker nodes is a peer-to-peer system. In peer-to-peer systems any node can exchange data with any other node. Here, every node performs a local anomaly detection and reports its results.

While peer-to-peer systems enable horizontal scalability, we need to deal with high CPU load and memory demands on the worker nodes. In addition, the learning of prototypes needs to be handled as there is no central instance. Either a central server needs to take over calculation of prototypes and therefore needs to gather monitoring information from worker nodes (see 3.4), or different prototype models are used on each worker node.

When using differing prototypes, anomalies depend on their local context. The local context is given by the results from neighbouring worker nodes e.g. worker nodes of the same rack. One batch job does not necessarily have to be an anomaly inside the system, when it is different from currently known prototypes on one single worker node. It might be consistent with prototype models on differing worker nodes. Therefore, before results are reported, the local context can be considered for local anomaly detection.

### *3.4. Hybrid peer-to-peer systems*

A hybrid peer-to-peer system additionally introduces a central server. As already mentioned for peer-to-peer systems, a central server can take over responsibility to calculate and distribute the prototype model. Besides this, a hybrid peer-to-peer system can be used to centralise anomaly detection on the central server. Worker nodes are then responsible for performing a preselection of data to reduce communication overhead. Again, the central server becomes a single point of failure and bottleneck of the system.

### *3.5. Super-peer systems*

A final architecture to consider is a super-peer system. Super-peers act as a centralised server to a subset of worker nodes. Different super-peers are connected to each other as peers. In addition, the subsets of worker nodes are also connected to each other as peers. Worker nodes perform preselection of data and report to their super-peer. The super-peers perform the actual anomaly detection for their associated set of worker nodes.

Super-peers are a temporarily used resource and therefore an interesting option for opportunistic resources. The already established monitoring of batch jobs allows the recognition of unsaturated worker nodes. Those include worker nodes that wait until all batch jobs are finished to e.g. perform software updates. The resources of such a worker node can be temporarily utilised as a super-peer. As soon as that state is dropped, data is lost and the associated worker nodes need to report their data to another super-peer.

## 4. Conclusion

To perform an online anomaly detection on payload level, a scalable architecture is required. Particularly given that batch jobs need to run uninterrupted even though distance measurements and anomaly detection require high computational cost and memory demands.

The most promising architecture combines elements from a hierarchical system as well as a super-peer system. A temporary utilisation of unsaturated worker nodes with additional central server components as fallback instances when resources need to be dropped brings a high flexibility. Hence, data consistency and coherence can be guaranteed. In addition, a modular software design that allows the distribution of different algorithmic steps such as preselection and anomaly detection, enables a good utilisation of that flexibility. On this way, variations such as anomaly detection based on local contexts can be explored.

## Acknowledgments

The authors would like to thank all people and institutions involved in the project Large Scale Data Management and Analysis, the German Helmholtz Association, and the Karlsruhe School of Elementary Particle and Astroparticle Physics for supporting and funding the work.

## References

[1] Kuehn E, Fischer M, Jung C, Petzold A and Streit A 2014 Monitoring data streams at process level in scientific big data batch clusters *BDC '14: Proc. of the 2014 IEEE/ACM Int. Symposium on Big Data Computing* ( IEEE Computer Society)
[2] Kuehn E, Fischer M, Giffels M, Jung C and Petzold A 2015 *J. Phys.: Conf. Ser.* **664** 052019
[3] Thain D, Tannenbaum T and Livny M 2005 *Concurr. Comput.: Pract. Exper.* **17** 323–356
[4] Chatrchyan S *et al.* (CMS) 2012 *Phys. Lett.* **B716** 30–61 (*Preprint* 1207.7235)
[5] Bille P 2005 *Theor. Comput. Sci.* **337** 217 – 239
[6] Pawlik M and Augsten N 2011 *Proc. VLDB Endow.* **5** 334–345